UNITED STATES PATENT APPLICATION

FOR

Method and Apparatus for Customized Logging in a Network Cache

INVENTORS:

Ashish Prakash
Robert English

Prepared by:

Blakely, Sokoloff, Taylor & Zafman LLP
12400 Wilshire Boulevard
Seventh Floor
Los Angeles, California 90025
(408) 720-8300

Attorney's Docket No. 5693P006X
(Client Docket No. P01-1294)

"Express Mail" mailing label number EL867650035US
Date of Deposit October 16, 2001
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

Julie Arango
(Typed or printed name of person mailing paper or fee)

Julie Arango 10/16/01
(Signature of person mailing paper or fee)

Method and Apparatus for Customized Logging in a Network Cache

[0001] This is a continuation-in-part of co-pending U.S. Patent application no. _____, filed on October 15, 2001, and entitled, "Method and Apparatus for Customized Logging in a Network Cache".

FIELD OF THE INVENTION

[0002] The present invention pertains to devices which proxy requests between clients and servers on a computer network. More particularly, the present invention relates to a method and apparatus for customized logging in a network cache.

BACKGROUND OF THE INVENTION

[0003] Of the many uses of the Internet, one of the more common ones is to access content on a remote server, such as a World Wide Web server. Typically, a person operates a client device to access content on a remote origin server over the Internet. The client may be, for example, a personal computer (PC) or a handheld device such as a personal digital assistant (PDA) or cellular telephone. The client often includes a software application known as a browser, which can provide this functionality. A person using the client typically operates the browser to locate and select content stored on the origin server, such as a web page or a multimedia file. In response to this user input, the browser sends a request for the content over the Internet to the origin server on which the content resides. In response, the origin server returns a response containing the requested content to the client, which outputs the content in the appropriate

manner (e.g., it displays the web page or plays the audio file). The request and response may be communicated using well-known protocols, such as transmission control protocol/Internet protocol (TCP/IP) and hypertext transfer protocol (HTTP).

[0004] For a variety of reasons, it may be desirable to place a device known as a proxy logically between the client and the origin server. For example, organizations often use a proxy to provide a barrier between clients on their local area networks (LANs) and external sites on the Internet by presenting only a single network address to the external sites for all clients. A proxy normally forwards requests it receives from clients to the applicable origin server and forwards responses it receives from origin servers to the appropriate client. A proxy may provide authentication, authorization and/or accounting (AAA) operations to allow the organization to control and monitor clients' access to content. A proxy may also act as (or facilitate the use of) a firewall to prevent unauthorized access to clients by parties outside the LAN. Proxies are often used in this manner by corporations when, for example, a corporation wishes to control and restrict access by its employees to content on the Internet and to restrict access by outsiders to its internal corporate network. This mode of using a proxy is sometimes called "forward proxying".

[0005] It is also common for a proxy to operate as a cache of content that resides on origin servers; such a device may be referred to as a "proxy cache". An example of such a device is the NetCache product designed and manufactured

by Network Appliance, Inc. of Sunnyvale, California. The main purpose of caching content is to reduce the latency associated with servicing content requests. By caching certain content locally, the proxy cache avoids the necessity of having to forward every content request over the network to the corresponding origin server and having to wait for a response. Instead, if the proxy cache receives a request for content which it has cached, it simply provides the requested content to the requesting client (subject to any required authentication and/or authorization) without involving the origin server.

[0006] Proxy caches may be used by corporations and other institutions in the forward proxying mode, as described above. Proxy caches are also commonly used by high-volume content providers to facilitate distribution of content from their origin servers to users in different countries or other geographic regions. This scenario is sometimes called "reverse proxying". As an example of reverse proxying, a content provider may maintain proxy caches in various different countries to speed up access to its content by users in those countries and to allow users in different countries to receive content in their native languages. In that scenario the content provider "pushes" content from its origin servers to its proxy caches, from which content is provided to clients upon request.

[0007] Customization of log files for a proxy cache allows administrators to define, collect, and process the information that is most useful in understanding typical web traffic patterns and anomalies. This information may also be used for billing purposes for users of the proxy cache.

[0008] Previous implementations provided a fixed set of pre-defined log file formats that were inflexible because the user did not have the choice to include or exclude a particular field from the log format. The user did not have the ability to create new fields for which to log information. The user also did not have the ability to specify the order of a particular field in a log file. What is needed, therefore, is a technique which overcomes these disadvantages of the prior art.

SUMMARY OF THE INVENTION

[0009] The present invention includes a method and apparatus for logging customized information in a network cache. The method comprises providing a user interface to allow a user to select a subset of a plurality of fields that may be present in a request for content from a client. The subset includes some or all of the fields and each field within the subset is printed to a log file in the sequence specified by the user.

[0010] Other features of the present invention will be apparent from the accompanying drawings and from the detailed description which follows.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which like references indicate similar elements and in which:

[0012] Figure 1 illustrates a network environment in which a proxy cache according to the present invention may be implemented;

[0013] Figure 2 illustrates an implementation of a logging system according to one embodiment;

[0014] Figure 3 illustrates a method by which data structures may be used within a logging system, according to one embodiment;

[0015] Figure 4 is a flow diagram showing the process of setting up the proxy cache for logging requests, according to one embodiment;

[0016] Figure 5 is a flow diagram illustrating the recording process according to one embodiment;

[0017] Figure 6 is a flow diagram showing the process of outputting information to a log file, according to one embodiment; and

[0018] Figure 7 is a block diagram showing an abstraction of the hardware components of the proxy cache, according to one embodiment.

DETAILED DESCRIPTION

[0019] A method and apparatus for customized logging in a network cache are described. Note that in this description, references to "one embodiment" or "an embodiment" mean that the feature being referred to is included in at least one embodiment of the present invention. Further, separate references to "one embodiment" in this description do not necessarily refer to the same embodiment; however, neither are such embodiments mutually exclusive, unless so stated and except as will be readily apparent to those skilled in the art. For example, a feature, structure, act, etc. described in one embodiment may also be included in other embodiments. Thus, the present invention can include a variety of combinations and/or integrations of the embodiments described herein.

[0020] As described in greater detail below, customized logging allows an administrator deploying a network cache to tailor information logged by the network cache. Information to be logged includes fields that are found in requests from clients and responses from servers. These fields are validated for correctness and the appropriate log control structures are initialized. The present invention uses a log control structure for each protocol to track fields to be logged and their relative positions in the log file. The following protocols may be supported by a proxy cache which uses this technique, for example: web access, streaming, streaming details, internet content adaptation protocol (ICAP), ICAP debug, network news transfer protocol (NNTP), and global request manager

(GRM). Each protocol has its own control structure, such that the log format for each protocol can be independently altered and customized at run-time. More specifically, the invention provides, for each protocol, a data structure that indicates whether or not each field in the protocol is to be logged and, if so, the order of that field within the log file. Changes to the log format can be made while the proxy cache is running, causing a new header to be written to the log file, thereby enabling administrators to alter the format on the fly. The algorithm for defining a log format determines whether all fields are valid and sets positional integers in the control structure when a new format is specified. As a transaction is processed, fields are checked to see if they should be logged by examining the control structure, and field data is stored in memory with relevant position and length information for each field. At the end of a transaction, every field that was selected is written to the log file in the correct order using the stored position and length information.

[0021] Note that the present invention can be used to provide customized logging of fields in either request from clients or responses from origin servers. To simplify description, however, the following description sometimes focuses on logging requests.

[0022] The framework allows for the addition of new fields and is easily maintainable using macros for field definitions and automatic generation of control structures and field literals. In order to detail the ability to add new fields, the operation of logging HTTP request and response headers will be

explored in further detail. There are HTTP headers (as defined by the Request for Comments (RFC) standard) that the application module of the proxy cache parses as part of its normal operation (i.e., looking for cacheability information, dates and times, etc.) and there are other headers including any proprietary headers that web servers and browsers may use. These headers will be referred to as "parsed" headers and "other" headers, respectively. A log format can include fields that represent both request and response headers. When an administrator enters a custom log format using the user interface of the proxy cache, the application module parses that log format to validate it and set up the log control structure (the structure that determines which fields are logged and their positions). This structure contains non-header fields. When the application module encounters a header field, it first determines whether it is a request or a response header. Then it determines whether it is a parsed header by searching through header parse tables, and if it is a parsed header it sets the positional field in a field in the parse table. If this is not a parsed header then the header and position is stored in a separate 'other headers' array.

[0023] Hence, in one embodiment header log positional information is stored in four different tables: 1) a table for parsed request headers, 2) a table for parsed response headers, 3) a table for 'other' request headers, and 4) a table for 'other' response headers. For parsed headers, logging information is stored in the appropriate parse table (request or response) and for other headers there are two separate arrays for other request headers and other response headers. When the

HTTP code is processing a request or a response and parsing headers it checks the parse tables and processes each header according to the parse table and also makes a call to the log module. If there is no flag present, that header is copied into the temporary log buffer and the appropriate position is noted. If this is another header and does not have an entry in the parse tables, the 'other headers' array is checked to see if there is a match. If there is a match, that header is logged.

[0024] As will become more apparent from the description which follows, the described technique allows network administrators to use log files for a variety of purposes, including monitoring and billing. Previous implementations allowed little control over choosing information written to log files, with only a few sets of pre-defined log file formats. Furthermore, allowing customization of the information being logged for each log file gives the user (e.g., administrator) control over exactly what data they can include in specific log files. In addition, the customized logging technique described herein complies with the WC3 standard, Hallam-Baker et al., "Extended Log File Format", World Wide Web Consortium (W3C) working draft WD-logfile-960323, which facilitates the use of log analysis software that conforms to that standard.

[0025] Another significant advantage of the proxy cache described herein is that the module which maintains knowledge of the various fields for each protocol (the "application module") is separate from, and independent of, the module which actually records the user-selected fields during a transaction (the

"logging module"). Consequently, new fields can be added and new protocols can be supported without any changes being required to the logging module.

[0026] Figure 1 illustrates a network environment in which a proxy cache according to the present invention may be implemented. As illustrated, a proxy cache 1 configured according to the present invention is connected between a LAN 2 and the Internet 3. A number (N) of clients 4-1 through 4-N are coupled to the LAN 2. A number (M) of origin servers 5-1 through 5-M are coupled to the Internet 3. The proxy cache 1 forwards requests from the clients 4 for content residing on the origin servers 5 and forwards content and/or other responses from the origin servers 5 to the appropriate clients 4. The proxy cache 1 also caches content from the origin servers 5. It may be assumed that the proxy cache 1 operates within a defined cache hierarchy.

[0027] Note that a proxy cache in accordance with the present invention can be used advantageously in network environments other than that shown in Figure 1. For example, a proxy cache according to present invention need not be used to couple clients on a LAN to the Internet. In other embodiments, one or more other types of networks may be substituted for either the LAN 2 or the Internet 3 in the configuration of Figure 1. Furthermore, a proxy cache may be used in either a forward proxying configuration or a reverse proxying configuration consistently with the present invention.

[0028] Figure 2 illustrates an example of the elements of the proxy cache 1, according to one embodiment. The proxy cache 1 includes an application

module 24 that receives requests for content on the network from a plurality of clients 4, receives corresponding responses from origin servers 5 (when requested content is not cached locally), and stores content (e.g., web pages or multimedia files) obtained for each client in a content cache 22. The application module 24 may be, for example, layer 7 software (with respect to the ISO/OSI model). These requests for content on the network and responses from origin servers each generally contain multiple fields that may be logged by the proxy cache 1. An administrator ("user") utilizes a user interface 23 to choose fields to be logged and the particular order in which the chosen fields are to be logged. A first data structure 31 in the proxy cache 1 is used to store indications of which fields the user has selected for logging and the order in which the user wants the chosen fields to appear in a log file.

[0029] Subsequently, in response to receiving a content request from a client or a response from a server, the application module 24 invokes a setup operation of setup and destroy module 26 in the proxy cache 1 to create a second data structure 32 and a third data structures 33, which are temporary data structures that will be used to record the information that is to be logged. The application module 24 calls the log module 25 for each user-selected field, which records the information in the second and third data structures 32 and 33. The output module 27 in the proxy cache 1 subsequently creates a log file 28 for the recorded field information. The log file 28 indicates the protocol and includes the logged fields in the user-selected order. Once logging has been completed, the log file 28

is saved to a non-volatile storage device, memory 29 in the proxy cache 1, and the second 32 and third 33 data structures are destroyed using a destroy operation of setup and destroy module 26.

[0030] Figure 3 shows an example of how to use data structures to implement the above described technique. A separate first data structure 31 is maintained in the proxy cache 1 for each protocol for which the proxy cache 1 supports logging. Hence, for each protocol, the corresponding first data structure 31 includes storage locations for all of the particular fields defined in that protocol. The first data structure 31 is initialized so that a predetermined flag (e.g., "-1") is placed in every location.

[0031] An administrator ("user") then utilizes the user interface 23 to select a protocol (e.g., HTTP, NNTP, or ICAP). The user then selects some or all of the fields defined within that protocol to be logged and the particular order in which the selected fields are to be logged. Once the fields and order are chosen by the user, the first data structure 31 for that protocol is modified so that the location, in the first data structure 31, of each field selected by the user for logging contains an integer value indicating the order in which the user wants that field to appear in the log file 28. The first data structure 31 in Figure 3 is illustrated to show just a few of the possible fields associated with the HTTP protocol (e.g., "c_ip, cs_uri", etc.), as an example. This example is discussed further below.

[0032] Subsequently, in response to receiving a content request or a response, the application module 24 checks the first data structure 31 for each of the fields

defined in the selected protocol to determine if the field has been selected for logging. If the user did not select that field, the application module 24 will see the predetermined flag (e.g., "-1") stored in the corresponding location of the first data structure 31 and, therefore, will not call the log module 25 to record that field; in that case, the application module 24 will go on to consider the next field defined in the protocol. If the user did select that field, however, the application module 24 will make a call to the log module 25, passing to it the integer value representing the user-specified order in which the field is to be logged, as stored in the first data structure 31. This action causes the log module 25 to obtain the information for that particular field from the request or response, record an ASCII representation of that information in the next available location of the second data structure 32 (note that formats other than ASCII may be used in alternative embodiments). The log module 25 also stores, in a location of the third data structure 33 corresponding to the user-selected order of that field, a reference to the field data stored in the second data structure 32. In one embodiment, the reference includes a position and length value of the ASCII field data in the second data structure 32.

[0033] Note that the second data structure 32 and third data structure 33 are created and destroyed on a transaction-by-transaction basis to avoid excessive memory usage. In contrast, the first data structure 31 persists between transactions.

[0034] Once all fields in the protocol have been either recorded (if present in

the request or response) or ignored in response to a transaction, the output module 27 writes the field information recorded in the second data structure 32 to a log file 28 in the order the user selected. This is done by sequentially accessing consecutive locations of the third data structure 33 to retrieve the references stored therein, to locate the data in the second data structure 32 in the correct order.

[0035] Among other advantages, this technique allows new fields to be added and new protocols to be supported without any changes being made to the log module 25. Only the application module 24 has specific knowledge of the fields defined for each protocol. Neither the log module 25 nor the output module 27 requires any knowledge (independent of the data structures) of the fields defined for any protocol, the particular fields selected for logging, or the order in which particular fields should be logged. During a transaction, the log module 25 simply records fields in the order of the calls made to it (for each field) by the application module 24; this order is arbitrary and is at the discretion of the system designer. However, when recording fields in the second data structure 32, the log module 25 also records a reference to the location of each recorded field in a location of the third data structure 33 that matches the user-specified order of that field, based on the value from the first data structure 31 that it received in the call. The contents of the third data structure 33 allow the recorded fields to be written to the log file 28 in the correct order. The order in which calls are made to the log module 25 and in which fields are recorded is

independent of the (user-specified) order in which the fields are written to, and appear in, the log file 28. Hence, new fields can be added and new protocols can be supported simply by updating the application module 24, without any changes being required to the logging module.

[0036] The customized logging technique can be described in terms of two phases. Phase one is during processing of a transaction where the application module 24 interacts with the log module 25 and each field is examined to see whether it has been selected for logging and, if it has, is copied into data the second structure (32) along with length and positional information. Phase two occurs at the end of a transaction, where the fields recorded in the second data structure 32 are written out in the correct order using the positional arguments stored in the third data structure 33.

[0037] Refer again to Figure 3, which shows an example based on HTTP. Prior to set up by the user, the values in the first data structure 31 are initially all set to -1, which is the predetermined flag. However, after the user has selected the protocol, the fields to be logged, and the order in which they are to appear in the log file 28, the value of each location in the first data structure 31 represents the user-specified order in which that field is to appear in the log file 28. Hence, the contents of the first data structure 31 shown in Figure 3 can be interpreted to mean that the user has specified fields "x-timestamp", "c-ip", "cs-uri", "x-transaction", and "bytes" to be logged for a request and to appear in that order in the log file 28. The flag values of -1 indicate that the fields "date", "time", and

"x_note" are not to be logged. Note that the user-specified sequence of logging is not dependent on the order in which the sequence values are physically stored in the first data structure 31.

[0038] Assume now that at run time, the proxy cache 1 receives a request, such as "GET http://www.yahoo.com/foo.gif HTTP/1.0". In response to the request, the application module 24 calls the setup and destroy module 26 to initialize the second data structure 32 and the third data structure 33. The application module 24 then selects a field for which there is a value other than -1 stored in the first data structure 31, and then calls the log module 25, passing to the log module 25 the user-specified (integer) sequence value stored for that field in the first data structure 31.

[0039] In the illustrated example, the first field to be recorded is the "bytes" field. So, the application module 24 calls the log module 25, passing to it the user-specified sequence number for the "bytes" field from the first data structure 31, which in this case is "4" (indicating the fifth position in the log file 28, since the sequence values are 0, 1, 2, 3, 4, ...). Accordingly, the log module 25 records the ASCII representation of the "bytes" field from the request starting from the first available byte of the second data structure 32, which in this case is byte 0. In conjunction with recording the "bytes" value in the second data structure 32, the log module 25 also records, in the third data structure 33, the position and length of the ASCII value as stored in the second data structure 32. The ASCII "bytes" value is specified to appear fifth in the sequence in the output log file 28.

Consequently, the log module 25 also writes the position value "0" into the fifth position of the third data structure 33, as shown in Figure 3. (The length of the ASCII string may also be stored at that location in the third data structure 33 but, to simplify explanation, is not shown in Figure 3.)

[0040] Next, the application module 24 calls the log module 25, passing to it the user-specified sequence number for the "cs_uri" field from the first data structure 31, which in this case is "2" (indicating the third position in the log file 28). Accordingly, the log module 25 records the ASCII representation of the "cs_uri" field starting from the next available location of the second data structure 32, which in this example is byte 10 of the second data structure 32. Because "cs_uri" is specified to appear third in the log sequence, the log module 25 also records the position value "10" (and length value) in the third position of the third data structure 33, as shown in Figure 3.

[0041] A similar process is carried out for the fields "x_timestemp", "x_transaction", and "c_ip", etc., in that order, until the transaction has been processed.

[0042] Next, phase two begins, in which the log module 25 examines the contents of the third data structure 33 sequentially, to write all of the fields recorded in the second data structure 32 to the log file 28 in the correct order.

[0043] Figure 4 is a flowchart of an embodiment of the process used to set up logging through the user interface 23. First, at block 41 the user selects the protocol under which information will be logged. The first data structure 31 is

initialized at block 42 by placing a predetermined flag (e.g., "-1") in each location. At block 43 the user then selects a subset of the fields available for logging under the selected protocol (the user can, of course, select all available fields). At block 44 the user selects the order of display in the log file 28, and at block 45 an integer value indicating the order of display of each selected field is placed in the location of the first data structure 31 corresponding with each selected field. Locations in the first data structure for fields that are not selected for logging remain set to the predetermined flag value.

[0044] Figure 5 is a flowchart showing an embodiment of the recording process. The recording process is performed in response to each transaction (request or response). The application module receives a request (or response) at block 50. The second and third data structures 32 and 33 are initialized at block 51. The application module then selects a first field defined in the protocol at block 52. For the selected field, at block 53 the application module 24 checks the first data structure 31 to see if the user selected that field for logging. If the user did not select that field (i.e., the predetermined flag is found at the location for that field in the first data structure 31), the application module 24 will not read the value of that field from the request and will go on to the next field at block 59. If the user did select that field, the application module 24 passes the user-specified sequence value of that field from the first data structure 31 to the log module 25. The log module 25 then obtains the information for that particular field from the request at block 54, converts the information to ASCII format at

block 55, records that information in the next available sequential location of the second data structure 32 at block 56, and determines (based on the integer value from the first data structure 31) the user-selected order for logging this field at block 57. At block 58 the log module 25 then stores, in a location of the third data structure 33 which corresponds to the selected order, a reference to the location where that field is recorded in the second data structure. This process is repeated during the transaction, per blocks 53, 59 and 60, until all fields defined in the protocol have been processed.

[0045] Figure 6 is a flowchart illustrating an embodiment of the output process of the invention. As with the recording process, the output process is also performed on a per transaction basis. Once the recording process shown in Figure 5 is complete (i.e., the values of all fields have been either stored in the second data structure or ignored), the output module 27 reads the reference stored in the first location of the third data structure 33 at block 61 and then reads the value stored in the second data structure 32 in the location specified by the reference in the third data structure 33 at block 62. The information contained in the referenced location of the second data structure is then written to the log file 28 at block 63. The next sequential location of the third data structure 33 is read at block 64, and the above process continues until all locations of the third data structure 33 have been sequentially accessed, such that all of the user-selected fields have been written to the log file 28 in the user-specified order. When the process is finished (per block 65), the log file 28 is

written to a non-volatile storage device 29 in the proxy cache 1, such as a disk drive, at block 66, and the second data structure 32 and third data structure 33 are destroyed at block 67.

[0046] Figure 7 is a block diagram showing an abstraction of the hardware components of the proxy cache 1, according to one embodiment. Note that there are many possible implementations represented by this abstraction, which will be readily appreciated by those skilled in the art given this description.

[0047] The illustrated system includes one or more processors 71, i.e. a central processing unit (CPU), read-only memory (ROM) 72, and random access memory (RAM) 73, which may be coupled to each other by a bus system 77 and/or by direct connections. The processor(s) 71 may be, or may include, one or more programmable general-purpose or special-purpose microprocessors, digital signal processors (DSPs), programmable controllers, application specific integrated circuits (ASICs), programmable logic devices (PLDs), or a combination of such devices. The bus system (if any) 77 includes one or more buses or other connections, which may be connected to each other through various bridges, controllers and/or adapters, such as are well-known in the art. For example, the bus system 77 may include a "system bus", which may be connected through one or more adapters to one or more expansion buses, such as a Peripheral Component Interconnect (PCI) bus, HyperTransport or industry standard architecture (ISA) bus, small computer system interface (SCSI) bus, universal serial bus (USB), or Institute of Electrical and Electronics Engineers

(IEEE) standard 1394 bus (sometimes referred to as "Firewire").

[0048] Also coupled to the bus system 77 are one or more mass storage devices 74, a network interface 75, and one or more input/output (I/O) devices 76. Each mass storage device 74 may be, or may include, any one or more devices suitable for storing large volumes of data in a non-volatile manner, such as a magnetic disk or tape, magneto-optical (MO) storage device, or any of various forms of Digital Versatile Disk (DVD) or CD-ROM based storage, or a combination thereof. RAM 73 and/or the mass storage device(s) 74 may be used to implement the content cache 26 (Figure 2).

[0049] The network interface 75 is one or more data communication devices suitable for enabling the processing system to communicate data with remote devices and systems via an external communication link 80. Each such data communication device may be, for example, an Ethernet adapter, a Digital Subscriber Line (DSL) modem, a cable modem, an Integrated Services Digital Network (ISDN) adapter, a satellite transceiver, or the like. Referring again to the embodiment of Figure 1, the network interface 75 is used by the proxy cache 1 to communicate both over the LAN 2 and over the Internet 3. In particular, the network interface 75 is the communications interface by which the proxy cache 1 receives and communicates requests and responses between clients and servers. In addition, the network interface 75 may also be the communications interface by which a user adds, modifies, or deletes forwarding rules used by the proxy cache 1. Note that while only one external communication link 80 is illustrated,

separate physical communication links may be provided for each network connection (e.g., to LAN 2, Internet 3), although that is not necessarily the case.

[0050] Since proxy cache 1 may be accessed by a user via network interface 75, proxy cache 1 does not necessarily require its own I/O devices 76. Nonetheless, such I/O devices may be included in some embodiments and may include, for example, a keyboard or keypad, a display device, and a pointing device (e.g., a mouse, trackball, or touchpad).

[0051] The above-described processes and techniques (e.g., request processing, customized logging, etc.) and the corresponding elements (Figure 2) may be implemented at least partially in software. Such software may be part of the operating system of the proxy cache 1. Such software may reside, either entirely or in part, in any of RAM 73, mass storage device(s) 74 and/or ROM 72. Such software may be executed by the processor(s) 71 to carry out the described processes and techniques.

[0052] Thus, a method and apparatus for customized logging in a cache have been described. Although the present invention has been described with reference to specific exemplary embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader spirit and scope of the invention as set forth in the claims. Accordingly, the specification and drawings are to be regarded in an illustrative sense rather than a restrictive sense.